Adaptive Probabilistic Flooding for Multipath Routing

Christophe Betoule[2], Thomas Bonald[1], Remi Clavier[2], Dario Rossi[1], Giuseppe Rossini[1,†], Gilles Thouenon[2]

[1] Telecom ParisTech, Paris, France
firstname.lastname@telecom-paristech.fr († corresponding author)
[2] Orange Labs, Lannion, France
firstname.lastname@orange-ftgroup.com

***Abstract*—In this work, we develop a distributed source routing algorithm for topology discovery suitable for ISP transport networks, that is however inspired by opportunistic algorithms used in ad hoc wireless networks. We propose a plug-and-play control plane, able to find multiple paths toward the same destination, and introduce a novel algorithm, called adaptive probabilistic flooding, to achieve this goal. By keeping a small amount of state in routers taking part in the discovery process, our technique significantly limits the amount of control messages exchanged with flooding – and, at the same time, it only minimally affects the quality of the discovered multiple path with respect to the optimal solution. Simple analytical bounds, confirmed by results gathered with extensive simulation on four realistic topologies, show our approach to be of high practical interest.**

I. Introduction

Reducing the overall power consumption of the Internet is a major challenge for future networking technologies. It is commonly accepted that a significant amount of power is consumed by the IP address look-up algorithm, which becomes prohibitive with the growing size of the routing tables. A viable option for telecom carriers and ISPs is to replace IGP routing by some appropriate level-2 technology, thus forming full mesh subnetworks at the IP level. While Ethernet is a natural candidate, it still requires highly dynamic switching tables; moreover, path discovery is based on flooding, which doesn't make it scalable.

In this paper, we focus on a novel level-2 architecture that relies on the following key principles:

- each source maintains a set of multiple distinct paths to each destination;
- the header of each packet consists of a sequence of labels, one per node on the corresponding selected path;
- each node receiving a packet (i) removes the first label of the header, if any, and forwards the packet accordingly (the node is a relay) (ii) sends it to the IP layer in the absence of label (the node is the destination).

Note that core nodes do *not* maintain routing tables, which avoid expensive address lookup algorithms. All routing information is contained in the packet itself. Paths are built and updated by the source nodes, which enables *multipath* routing. For instance, each source maintains a primary path to each destination (typically, the shortest path), as well as a secondary path in case of failure or traffic surge. Unlike traditional IP routing, this secondary path is always available, which enables fast restoration in case of failure.

A critical component of this architecture is the algorithm used to discover paths. We propose a novel flooding algorithm inspired by the opportunistic algorithms of peer-to-peer applications, which consists in adapting the flooding rate of core nodes to the number of already received discovery messages. We refer to this algorithm as *adaptive probabilistic flooding*. At the cost of a limited number of state variables in core routers, the algorithm is able to discover multiple paths in a quasi-optimal way. Using analytical bounds, we prove that, unlike pure flooding, the algorithm scales with the network size.

The rest of the paper is organized as follows. Related work is presented in the next section. The path discovery algorithm is described in Section III. Section IV is devoted to the performance analysis. Section V concludes the paper.

II. Related work

Routing is a critical component of the Internet, and as such has long been studied by the scientific community. The problem of finding a *single path* interconnecting any two nodes of a graph is solved by well-known algorithms like Dijkstra and Bellman-Ford, which have been implemented in widely deployed protocols such as OSPF and RIP, respectively. However, interconnecting nodes through a single path (typically, the shortest) does not make the network resilient against failures and traffic surges. Hence, different techniques relying on *multiple paths* have been proposed. For instance, ECMP [1] aims at balancing load over multiple paths of equal cost. In standard IP/MPLS networks, the control and data planes are generally considered jointly; multipath routing is then achieved through a centralized algorithm, solving some standard multicommodity flow problem [2], [3].

In this work, we focus on the control plane and address the issue of the efficient discovery of multiple paths, as in [4]–[11]. Some of the above papers consider the problem of finding a pair of disjoint paths between any two nodes within the network, considering that the primary (shortest) path is *already known*. In [4]–[6], the problem is shown to boil down to a standard shortest path problem by some appropriate modification of the original graph. Ogier, Rutenburg and Shacham define

in [4] an algorithm that achieves such a graph modification and evaluates its performance in terms of communication, convergence time, and space complexity. Sidhu, Nair and Abdallah enhance this algorithm in [6] by finding all possible disjoint paths between any two nodes of the network. Eppstein proposes in [5] an algorithm that finds the first k shortest paths between any two nodes by a breadth first search of a 4-heap in which every node represents a path. Finally, works like [7] take a more practical approach, and enhance IP routing by means of centralized algorithms to determine disjoint paths, and distributing such paths through the routers taking care of avoiding loops.

The problem of finding multiple paths *without* any a priori knowledge such as the shortest path is quite different. Most algorithms then rely on flooding [8]–[11]. A swarm intelligence based solution is proposed with the Ant Colony Optimization (ACO) algorithm [9], where a set of ants is spread through the network in order to discover disjoint multiple paths: the pheromone left by the ants is employed in order to avoid already crossed paths. In [10], a flooding algorithm on layered routing architecture is employed: the basic idea is to give a score to each packet and to decrease this score for each link on which the packet is flooded; only nodes along the best path can increase the score and re-flood the packet. Authors in [11] propose a strategy where a scout message walks through the network accumulating nodes discovered in the walk: nodes re-flood the message only when the current discovered path differs significantly from the stored shortest path, allowing thereby to find multiple paths. In a very different context, namely ad-hoc wireless networks, flooding-based technique are exploited by Dynamic Source Routing (DSR) [8] where however multiple paths are not taken into account.

To the best of our knowledge, our approach introduces a number of novel ingredients. Similarly to DSR [8], in our approach the source inserts one label per node on the path to the destination, and the data plane forwards the packet by popping a label from the packet header at each hop. Unlike DSR, we don't have collisions, or problems inherent to wireless networks, yielding to a radically different algorithm design. The adaptive probabilistic algorithm we propose greatly limits the number of exchanged messages, as in [10], without however requiring packets to carry the scores associated with the discovery algorithm. The limited amount of state kept by nodes is not used to avoid crossing already traveled paths as in [9], but rather to avoid taking these paths too often, which has important consequences on the quality of the multiple paths discovered. Finally, unlike [11], the algorithm does not rely on threshold-based decisions, nor it depends on topological properties of the network – rather, its design makes it robust and auto-terminating irrespectively of its actual parameter setting, with performance that degrades gracefully in case of parameter misguidance.

III. Path discovery algorithm

A. Overview

We aim at designing a distributed algorithm for path discovery, capable of finding multiple, possibly disjoint paths between any pairs of nodes. To do so, each node periodically advertises its presence by means of some flooding procedure described below. Specifically, each node sends an *advertisement* message every τ_a seconds; typical values range from a few seconds to minutes [3]. Besides, each node sends keep-alive messages over each path in order to ensure that this path has not failed; the corresponding *refresh* messages are sent every τ_r secondes, typically set to a few milliseconds [12].

During the flooding procedure, each relay node adds its identifier to the advertisement messages it receives, so that these messages carry information concerning the whole traveled path. Upon reception of an advertisement message, a node learns a path from the source of this message, as well as from any intermediate node on this path, as in DSR [8]. Flooding decisions are taken independently by each node, and consitute the core of the algorithm. The main idea is that nodes need to flood a received message *at least once*, so that shortest paths are discovered. Nodes actually need to flood the message *multiple times*, in order to discover further paths beyond the shortest one. The number of flooding decisions is critical with respect to both the quality of the path discovery and the overhead of the algorithm.

A simple option could consist in including a Time To Leave (TTL) field in the packet, so as to interrupt the flooding process when some pre-configured maximum path length is reached. The selection of a proper TTL value is critical in this case: if the TTL is shorter than the graph diameter D for instance, then connectivity cannot be guaranteed; if the TTL is too large, the overhead of the algorithm becomes prohibitive (as the number of relayed messages is exponential in the TTL).

We propose an alternative approach based on *adaptive probabilistic flooding*. Any node receiving some advertisement message from source s floods this message the first time, and floods it with some decreasing probability the following times. Specifically, node i floods an advertisement message generated by source node s over all its links (except the one from which it has received the message) with probability:

$$P = \beta^{n_{i,s}} \tag{1}$$

where β is some fixed parameter and $n_{i,s}$ is a counter, stored at node i, of the number of times node i has already received an advertisement originated by node s. The flooding decision are taken independently on each link, and the counter is reset periodically, as explained later. Note that node i floods the first advertisement message it receives for source node s since $n_{i,s} = 0$ in this case. As further messages are received, flooding will become exponentially less likely, according to the backoff parameter β. The quality of the path discovery is expected to increase with β, at the expense of larger overhead. However, we shall see that performance is not very sensitive

to this parameter, which makes the algorithm robust and pratically interesting.

B. Primary and secondary paths

Consider a network, modeled as an undirected graph $G = (E, V)$, composed of $|V| = N$ routers, in which any pair of adjacent routers are connected by a single link for simplicity (the algorithm can be easily extended to the general case of multiple links between any pair of nodes). Between any two routers $i, j \in V$, we are interested in finding a pair of *paths*, i.e., sequences of edges connecting node i to j. We denote by $\mathcal{P}$ and $\mathcal{S}$ the primary and secondary paths, respectively, returned by the adaptive probabilistic algorithm on graph G, as described below. We denote by L_p and L_s the respective lengths of these paths.

To gauge the quality of the primary and secondary paths found by our algorithm, we need to define target path properties. The primary path is expected to be the shortest path in number of hops; in other words, we say that $\mathcal{P}$ is optimal if it belongs to the set of shortest paths from i to j in G (as there may be several such paths). The secondary path is expected to minimize the similarity with the primary path, $\mathcal{P} \cap \mathcal{S}$. Note that this choice reduces the share of faith between these paths, improving network resilience against failures and traffic surges.

To find the optimal secondary path $\mathcal{S}$, we consider a modified graph G' in which the cost of links along the primary path $\mathcal{P}$ are increased by the network diameter [4], and other link costs are unitary. As links belonging to $\mathcal{P}$ are now discarded due to higher cost, running Dijkstra on G' we retrieve a path $\mathcal{S}'$ minimizing the similarity function $\mathcal{P} \cap \mathcal{S}'$ (notice that since nodes along the primary path are not removed from G', they can be included in $\mathcal{S}'$ only if strictly necessary as the path would otherwise be disconnected). We say that the secondary path found by the algorithm $\mathcal{S}$ is optimal if $|\mathcal{P} \cap \mathcal{S}| = |\mathcal{P} \cap \mathcal{S}'|$ and $L_s = L_{s'}$, i.e., the length L_s of the secondary path is equal to the length $L_{s'}$ of the optimal $\mathcal{S}'$ (as there may be multiple disjoint paths minimizing the similarity with the shortest path).

C. Metrics

To precisely quantify the *overhead* vs *path quality* tradeoff early outlined, we resort to the following metrics. The average amount of messages handled by any given node during the advertisement procedure is denoted with M_a. For each path, the refresh process then requires each node to send keep-alive messages: we denote by M_r the average number of such messages (counted once per every link traveled). The relative weigth of $M_a/(M_r + M_a)$ quantifies the *overhead* induced by the advertisement process.

In the following, we evaluate the algorithm in terms of *connectivity* along the primary and secondary path (i.e., whether paths $\mathcal{P}$ and $\mathcal{S}$ joining any two nodes $i, j \in V$ exist) and *optimality* (i.e., whether $\mathcal{P}$ and $\mathcal{S}$ are optimal according to the above definitions). We express connectivity in terms of the probability C_p (respectively, C_s) that, $\forall i, j \in V$, nodes i and j are connected by some primary (respectively, secondary) path. We express optimality in terms of the probability O_p (respectively, O_s) that the primary path is also the shortest (respectively, that the secondary path is the shortest, most diverse path).

```
1: while {receiving message ADV} do
2:    ℓ ← length(ADV.ID)
3:    for all { i ∈ [0, ℓ] } do
4:       if {ADV.ID[i] = j} then
5:          exit // Break loop and abort flooding
6:       else
7:          d ← ADV.ID[i] // Destination
8:          L_{j,d} ← (ADV.ID[ℓ], ..., ADV.ID[i])
            // Overhearing advertised paths from ADV;
9:          if { ∄P_{j,d} ∨ length(L_{j,d}) < length(P_{j,d})}
            then
10:            P_{j,d} ← L_{j,d} // Update primary path
11:         end if
12:         if {∄S_{j,d} ∨ |P_{j,d} ∩ L_{j,d}| < |P_{j,d} ∩ S_{j,d}| ∨
            (|P_{j,d} ∩ L_{j,d}| = |P_{j,d} ∩ S_{j,d}| ∧ length(L_{j,d}) <
            length(S_{j,d}))} then
13:            S_{j,d} ← L_{j,d} // Update secondary path
14:         end if
15:      end if
16:   end for
17:   append j to ADV.ID
18:   s ← ADV.ID[0] // Source
19:   for all {next ∈ neighbors(j) } do
20:      if { next ≠ ADV.ID[ℓ − 1] } then
21:         send ADV to next w.p. β^{n_s}
            // Adaptive probabilistic flooding
22:      end if
23:   end for
24:   n_s++ // Update counter associated with source s
25: end while
```

Fig. 1. Algorithm pseudo-code for a generic node j of the network

D. Pseudocode

A pseudocode description of the algorithm is given in Fig. 1. A source node s initiates the advertisement process by flooding an advertisement packet ADV to all its neigbors. The flooded packet contains a list of node identifiers ID, initially set to ID[0]=s by the source, to which each node appends its own identifier. Upon reception of an advertisement packet ADV, a node learns a (backward) path to the source s and to any intermediate node $d = \text{ADV}.ID[i]$ along the path. In case the receiver j detects a loop (finding its identifier within the ID list), it discards the message and aborts the flooding procedure. Otherwise, it analyzes, and possibly stores, the newly learned path $\mathcal{O}_{j,d}$. Specifically, the primary (and secondary) path is first set if not existent yet. Also, if the newly overheard path is shorter than the primary path $length(\mathcal{L}_{j,d}) < length(\mathcal{P}_{j,d})$, then the primary path is updated with the overheard one. Similarly, if the overheard path has lower similarity than the current secondary path $|\mathcal{P}_{j,d} \cap \mathcal{L}_{j,d}| < |\mathcal{P}_{j,d} \cap \mathcal{S}_{j,d}|$, or if it has equal similarity but is shorter than the secondary $|\mathcal{P}_{j,d} \cap \mathcal{L}_{j,d}| = |\mathcal{P}_{j,d} \cap \mathcal{S}_{j,d}| \wedge length(\mathcal{L}_{j,d}) < length(\mathcal{S}_{j,d})$, then the secondary path is updated.

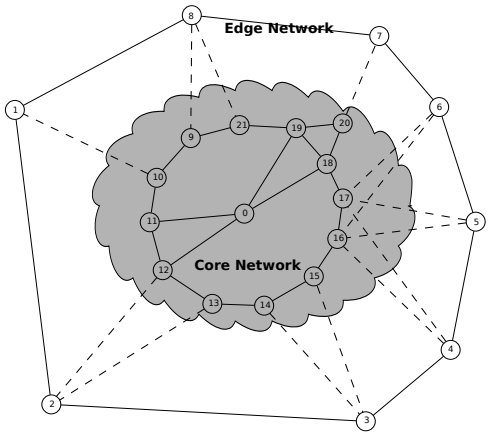


Fig. 2. Tiger2: an example of hierarchical AS network. The core network is highlighted by the grey balloon, and router 0 represents the collect point toward the Internet core.

Notice that we expect messages on the shortest path to reach a node *before* messages that take longer paths: in case of homogeneous setup (i.e., equal latencies) this invariant always holds. As such, the shortest path is chosen relatively early in the advertisement process and is typically never changed later on, as its change would also affect the quality of the secondary path. Not shown in the pseudocode for the sake of clarity, ties in the secondary path selection are broken at random.

Finally, after having added its own identifier to the ADV.ID, the node probabilistically floods the ADV message, with independent decisions per each neighbor (except the node ADV.ID[ℓ-1] from which the message came), and update the per-source counter n_s. As we shall see, the duration of each discovery phase is much shorter than the discovery period τ_a, which implies that the counter n_s can be safely reset for new discoveries.

IV. Performance evaluation

In this section, we first evaluate the overhead of the algorithm through a simple analytical model, and then employ discrete event simulation to validate the analysis and evaluate performance in terms of the quality of discovered paths.

Simulations are carried on with Omnet++ [13] over four different network topologies, whose most significant properties are summarized in Tab. I. Specifically, Tab. I reports the number of nodes N, the average and standard deviation of the node degree, $\bar{\delta}$ and σ, the average length of the optimal primary $\overline{L_p}$ and secondary $\overline{L_s}$ paths, the diameter D of the original grap G and the largest diameter D' over the modified G' graphs.

Note that we consider both real network topologies (Tiger2 [14], Abilene [15], Geant [16]), corresponding to different segments, as well as a set of 50 synthetic random graphs (Random) whose number of nodes and degree distribution loosely fit the real topologies (in case of the Random topology, D and D' are averaged over the 50 considered instances). All network topologies are well known except for Tiger2, that we depict in Fig. 2. For the time being, we use homogeneous settings (i.e., constant and equal delay on every link), and no failures happen within the network. The evaluation of more complex (heterogeneous delay, failures, etc.) scenarios is part of our ongoing work.

TABLE I
Topological properties of the network scenarios

Network	Segment	N	$\bar{\delta}$	σ	$\overline{L_p}$	$\overline{L_s}$	D	D'
Tiger2 [14]	metro	22	3.6	0.6	2.7	3.7	5	6
Geant [16]	aggregation	22	3.4	1.4	2.6	4.0	6	10
Abilene [15]	core	11	2.6	0.5	2.4	4.2	5	8
Random	synthetic	22	3.2	1.6	2.6	3.8	5.4	7.3

A. Overhead

Pure flooding generates a number of messages that is exponential in the number of nodes N. Our adaptive probabilistic flooding algorithm is scalable in the sense that it generates only $O(N)$ messages, as shown below, at the expense of $O(N)$ counters, cf. (1).

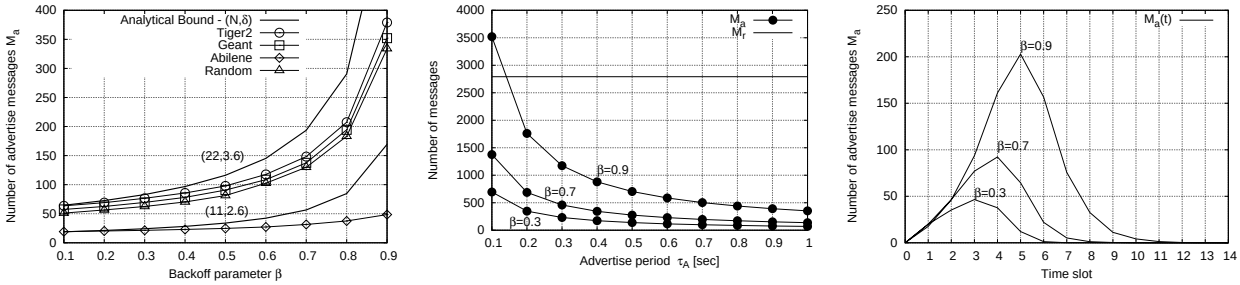
Consider a single advertisement from some source node s, and consider some relay node j with degree δ. The first time node j receives a ADV message originated by s, it sends a copy on each output link, except the one where the ADV message has been received. This generates $\delta - 1$ messages. The second time j receives a ADV message from the same source, it will forward the message over each of the $\delta - 1$ links with probability $\beta < 1$; so, at second reception, node j generates $(\delta - 1)\beta$ messages on average. Iterating and taking into account N advertisement processes (one per each node), we bound the total number of control messages M_a that are seen by the average node:

$$\begin{aligned} M_a &\leq N[(\bar{\delta}-1) + (\bar{\delta}-1)\beta + (\bar{\delta}-1)\beta^2 + \cdots] \\ &= N(\bar{\delta}-1)\sum_{n=0}^{\infty}\beta^n \\ &= N(\bar{\delta}-1)\frac{1}{1-\beta} \end{aligned} \quad (2)$$

(in other words, the whole network will carry NM_a messages for a full round of N advertisement processes). Note that (2) is an upper bound since we do not account for the detection of loops, which reduces the actual number of transmitted messages. While it is in principle possible to refine the bound by considering the probability that loops form, this can be done in closed form only for simple topologies such as random graphs [17]. Moreover, it turns out that this simple and conservative bound matches very well, as we will see, the empirical results found by simulation.

Note also that, as the first flood is always performed, convergence of the primary $\mathcal{P}$ path to the shortest path is always guaranteed. Hence, the backoff parameter β affects only the quality of the secondary path $\mathcal{S}$: by tuning β, we can upper bound the algorithm overhead M_a while matching the required level of path quality.

Now let us focus on the number of messages generated during the advertisement process. Fig. 3(a) depicts, as a



(a) Number of advertisement messages generated by a full discovery round of all the nodes in the network as a function of β

(b) Number of refresh M_r and avertisemed M_a messages generated during a fixed time window $W = 1$ sec as a function of the advertisement period τ_a (Tiger2 network)

(c) Time evolution of the number of avertisement messages (Tiger2 network)

Fig. 3. Overhead generated by adaptive probabilistic flooding.

function of β, the upper bound (2) along with the number of messages M_a gathered by means of simulation, for the four topologies early outlined. Note that the upper bound associated with Abilene network is computed using the fitted parameter values $(N, \bar{\delta}) = (11, 2.6)$, while for the other 22-nodes networks, only the upper bound for the Tiger2 topology is shown $(N, \bar{\delta}) = (22, 3.6)$ to avoid cluttering the picture. It can be noted that the overall number of messages generated by the advertisement procedure triggered by all nodes in the network remains low, topping to a few hundreds for high values of β. Also, notice that the number of messages generated in all the real networks is very similar, with Tiger2 as the worst case. This shows the robustness of our algorithm, given the various topological properties of the considered networks.

Although the overall number of messages remains low for any value of β, we miss a reference for comparison: more insights can be gathered from Fig. 3(b), that compares the number of messages generated by the advertise procedure with the periodic keep alive messages due to the path refresh function. Considering a fixed observation time window of $W = 1$ s, we vary the τ_a period in $[0.1, 1]$ s interval and set the refresh time to the typical value of τ_r=50 ms. We then compute the overall number of refresh messages M_r by taking into account the actual primary $\mathcal{P}$ and secondary $\mathcal{S}$ paths lenghts gathered in simulation as $M_r = \frac{W}{\tau_r}(length(\mathcal{P}) + length(\mathcal{S}))$. We finally gauge the number of advertisement messages in simulation for the Tiger2 network, that represents the worst case, so as to gather conservative results. As the picture clearly shows, the overhead due to the advertise procedure is very low even for very small durations of the advertise period τ_a (well below 1 sec), and even for large values of β: for $\tau_a \geq 1$ sec the overhead becomes clearly negligible for any $\beta \leq 0.9$.

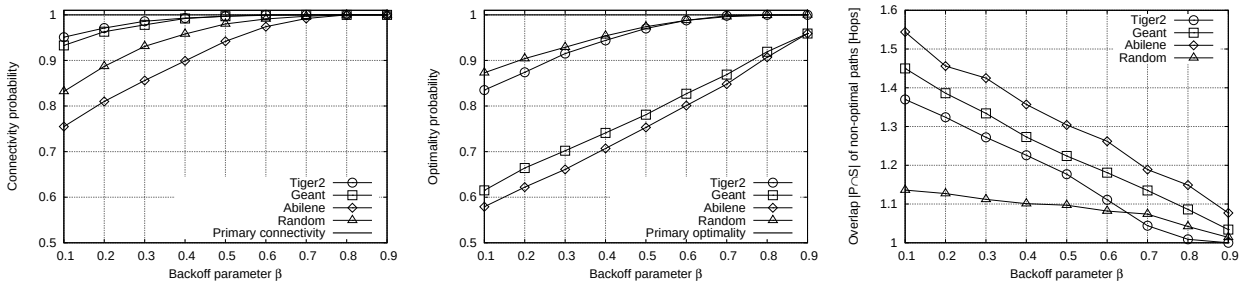
Finally, Fig. 3(c) depicts the evolution over (slotted) time of the number of messages carried over the Tiger2 network during an advertisement originated by a single source: we observe that, after some initial exponential growth due to the flooding process, the backoff factor kicks in and slows down the growth, which then dies out very fast, due to an increasing number of flooding paths being probabilistically cut out. This auto-termination feature is a very desirable property of the algorithm, and further suggests that advertisement periods do not need to overlap, but can rather be *staggered*. This could be achieved either with a simple policy (e.g., periodically at random within $[0, 2\tau_a]$) or with more sophisticated schemes (e.g., each node deciding autonomously whether to trigger a new advertisement depending on the measured control messages load). In turn, this implies that instead of keeping $O(N)$ counters (one for each source in case of advertisements *in parallel*), the system could perform advertisement *in series* and keep a small number of $O(1)$ counters (using a modulo function to solve unlikely contentions due to independent simultaneous triggering of advertisement processes by multiple nodes). This is an interesting direction for future research, that we aim at pursuing in the following.

B. Path quality

Let us now focus on the quality of the paths that the adaptive probabilistic flooding algorithm is able to find. For simplicity, we let each node advertise itself once at time $t = 0$ and evaluate the connectivity and optimality of the primary and secondary paths. Results are averaged over 10 simulations over the real topologies, and over 50 graph instances in the synthetic Random graph case.

Fig. 4(a) depicts the *connectivity* probability of the primary and secondary paths as a function of β: as expected, primary connectivity does not depend on β and is always guaranteed. Since a primary path is always found, the connectivity index is relevant for the secondary path only: we see that all secondary paths are connected in all networks when $\beta \geq 0.7$ (which correspond to small overhead in Fig. 3(b)).

Fig. 4(b) reports the *optimality* probability of the primary and secondary paths as a function of β: again, since the shortest path is always eventually found, the optimality of the primary path is guaranteed. Thus, the optimality index is relevant only for the secondary path: we see that a significant percentage (from 60% to 85%, depending on the topology) of secondary paths are optimal even for a very low value of $\beta = 0.1$, and that at least 90% of secondary paths are optimal for all considered topologies when $\beta \geq 0.8$. Moreover, we



(a) Connectivity probability of primary and secondary paths

(b) Optimality probability of primary and secondary paths

(c) Average overlap between primary and secondary path $||\mathcal{P} \cap \mathcal{S}||$ for non-optimal seconday paths overlap

Fig. 4. Quality of the paths found by adaptive probabilistic flooding.

observe that optimality gracefully degrades β, and furthermore with similar (roughly linear) slope across all topologies. This is a desirable behavior: as no phase transition nor knee appear in the path quality slopes, tuning β between low overhead (low β) vs high path quality (high β) is not critical.

Finally, we dissect the reason behind the sub-optimality of some secondary paths. Recall that a secondary path is optimal if it is the shortest and most diverse path compared to the primary. Hence, sub-optimality of the secondary path may be due to either (i) a non-zero *overlap* between primary and secondary paths, $|\mathcal{P} \cap \mathcal{S}| > 0$, or (ii) a path with a stretch over the optimal secondary path larger than one $L_s/L_{s'} > 1$. Fig. 4(c) depicts the overlap, i.e., the number of nodes that primary and secondary paths have in common, conditioning over the sub-optimal paths (i.e., the overlap of optimal secondary paths is not accounted for in the picture). As shown by the figure, sub-optimality seems to be tied to slightly more than one node in common as $|\mathcal{P} \cap \mathcal{S}| \in [1, 1.5]$. Furthermore, as the average overlap is always $|\mathcal{P} \cap \mathcal{S}| \geq 1$ for any β, we can conclude that overlapping paths are significantly more common that long-stretching paths.

V. Conclusions

We have presented a novel flooding based algorithm for multiple-path discovery: the algorithm trades a small amount of state in routers, i.e., $O(N)$ counters, in order to significantly limit the number of messages generated by flooding through an adaptive probabilistic algorithm.

Simple analytical bounds, confirmed by simulation results, show the overhead entailed by the advertisement procedure to be low (with respect to the amount of keep-alive messages needed to keep the path up-to-date) and auto-terminating (due to the multiplicative decrease of the flooding probability).

Simulation results also testify excellent performance in terms of path quality: connectivity and optimality of the primary path are achieved by design, while 90% of secondary paths are also optimal when $\beta \geq 0.8$ (or otherwise decrease linearly for lower β). Interestingly, the low percentage of low path is due to a very limited amount of share of faith between paths (1.5 nodes in the worst case, for $\beta = 0.1$).

As part of our future work, we want to carry on more realistic experiments on a wider set of topologies and further reduce the amount of state to $O(1)$.

VI. Acknowledgement*

This work was funded by Celtic TIGER2 project.

References

[1] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992 (Informational), Nov. 2000.

[2] R.A. Guerin, A. Orda, and D. Williams, "Qos routing mechanisms and ospf extensions," in *Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE*, Nov. 1997, vol. 3, pp. 1903 –1908 vol.3.

[3] Srihari Nelakuditi and Zhi-Li Zhang, "On selection of paths for multipath routing," in *Quality of Service IWQoS 2001*, Lars Wolf, David Hutchison, and Ralf Steinmetz, Eds., vol. 2092 of *Lecture Notes in Computer Science*, pp. 170–184. Springer Berlin / Heidelberg, 2001.

[4] R.G. Ogier, V. Rutenburg, and N. Shacham, "Distributed algorithms for computing shortest pairs of disjoint paths," *Information Theory, IEEE Transactions on*, vol. 39, no. 2, pp. 443 –455, Mar. 1993.

[5] D. Eppstein, "Finding the k shortest paths," *Foundations of Computer Science, Annual IEEE Symposium on*, vol. 0, pp. 154–165, 1994.

[6] Deepinder Sidhu, Raj Nair, and Shukri Abdallah, "Finding disjoint paths in networks," *SIGCOMM Comput. Commun. Rev.*, vol. 21, pp. 43–51, August 1991.

[7] P. Merindol, J.-J. Pansiot, and S. Cateloin, "Low complexity link state multipath routing," in *INFOCOM Workshops 2009, IEEE*, april 2009, pp. 1 –6.

[8] D. Johnson, Y. Hu, and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," RFC 4728 (Experimental), Feb. 2007.

[9] Maria J. Blesa and Christian Blum, "Ant colony optimization for the maximum edge-disjoint paths problem," in *EvoWorkshops*, 2004, pp. 160–169.

[10] Tein-Yaw Chung Yung-Mu Chen and Yang-Hui Chang, "Simple qos routing algorithms on layered routing architecture," in *Proc. Active Networking Workshop 2002 (ANW '02)*, 2002.

[11] J. Chen, P. Druschel, and D. Subramanian, "An efficient multipath forwarding method," in *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, mar-2 apr 1998, vol. 3, pp. 1418 –1425 vol.3.

[12] K. Fouli and M. Maier, "The road to carrier-grade ethernet," *Communications Magazine, IEEE*, vol. 47, no. 3, pp. S30 –S38, march 2009.

[13] Andras Varga, "Omnet++ website," 2010.

[14] "Celtic tiger2 project website," http://projects.celtic-initiative.org/tiger2/.

[15] "Internet2 network website," http://www.internet2.edu/network/.

[16] "Geant project website," http://www.geant.net/pages/home.aspx.

[17] Svante Janson, "The numbers of spanning trees, hamilton cycles and perfect matchings in a random graph," *Combinatorics, Probability and Computing*, vol. 3, pp. 97–126, 1994.